LEVEL II

NRL Report 8554

Best Available Technologies (BATs) for Computer Security

CARL E. LANDWEHR

Computer Science and Systems Branch
Information Technology Division

December 21, 1981

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8554	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BEST AVAILABLE TECHNOLOGIES (BATs) FOR COMPUTER SECURITY		5. TYPE OF REPORT & PERIOD COVERED Interim report on a continuing NRL problem. 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Carl E. Landwehr		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N; RR0140941; NRL Problem 75-0113-0-1
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE December 21, 1981 13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer security	Capability architectures	Security kernel
Program verification	Formal specification	Risk assessment
Security measures	Security models	Virtual machines
Virtual memory	Software engineering	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Over more than a decade, government, industry, and academic centers have invested substantial resources in techniques for developing secure computer systems. The major projects to develop such systems are surveyed from the perspective of a system developer about to undertake the design and implementation of a secure software system. Experience with specific techniques and applications is summarized, and recommendations are given. Suggested approaches to each phase of the system life cycle are provided. The principal recommendations to developers are that they consider the security requirements of each system as part of its functional requirements, rather than as a

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICAT OF THIS PAGE (When Data Entered)

20 ABSTRACT (Continued)

separate set of requirements; that they continue to think about security throughout the design and implementation of the system; and that they use the best available software engineering technology.

CONTENTS

INTRODUCTION 1

ASSUMPTIONS 1

CASES OF INTEREST 2

EXPERIENCE WITH TECHNIQUES AND APPLICATIONS 2

ADVICE FOR THE DEVELOPER 10

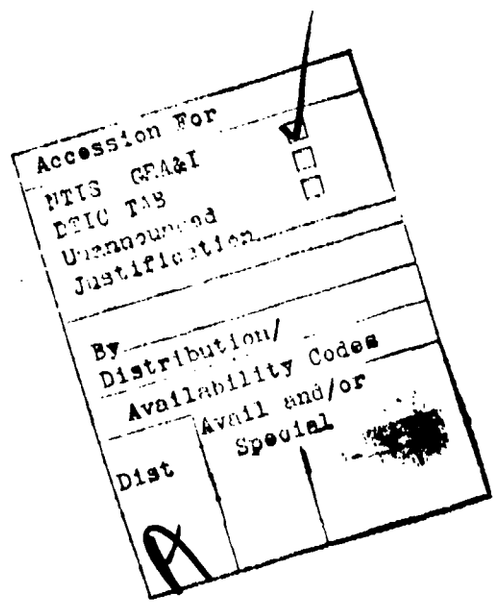
SUMMARY 14

ACKNOWLEDGMENTS 14

REFERENCES 14

APPENDIX A—Comments on Secure-System Developments 15

APPENDIX B—Bibliography 21



BEST AVAILABLE TECHNOLOGIES (BATs) FOR COMPUTER SECURITY

INTRODUCTION

Over more than a decade, government, industry, and academic centers have invested substantial resources in techniques for developing secure computer systems. A good deal has been learned about the problem, and a number of approaches have been explored. But what useful advice can we give to those about to embark on a system development? If the application requires a multilevel secure system, how should the developer proceed? The purpose of this report is to summarize past experience and to guide developers.

The next section lists a few general assumptions about the system developer's environment. The third section describes past, present, and planned projects that have attempted to build secure computer systems. A brief discussion of each system listed in the tables can be found in Appendix A. The fourth section summarizes the experience gained in developing secure systems by listing a number of techniques and applications (alphabetically by "buzzword") and providing a brief discussion of each. The fifth section provides advice to developers, organized by the phases of the system development cycle.

There is no simple recipe for creating a multilevel secure system. The "three-layer" security kernel approach, in which application programs execute on an operating system emulator that interfaces to the security kernel, has yet to produce a system that is efficient enough for practical use (although there is some hope that kernels tailored to specific applications may yet achieve this goal). There has been considerable work on formal specification and program verification techniques, but the tools available to support these activities require further development before they can be employed in development efforts.

To be secure, a computer system must reliably enforce a specified policy for accessing the data it processes while it accomplishes the functions for which it was built. Since software engineering has as its goal the production of reliable, maintainable programs that perform according to their specifications, the best techniques developed for software engineering should be the cornerstone of efforts to develop secure computer systems.

The principal recommendations to developers are (1) that they consider the security requirements of each system as part of its functional requirements rather than as a separate set of requirements; (2) that they continue to think about security throughout the design and implementation of the system; and (3) that they use the best available software engineering technology [1].

ASSUMPTIONS

We assume that the developer is to construct a system for some particular application (e.g., packet switching, message processing, software development) and that he has substantial control over the choice of hardware for the system, the choice of software design and documentation strategy, and the choice of validation and verification strategy. He will have somewhat less control over the choice of programming language; he will have much less control over the interfaces required of his system; and he will have still less control over the set of systems with which the new system is required to interface.

Manuscript submitted September 18, 1981.

CASES OF INTEREST

We choose not to list all of the journal articles, memoranda, research papers, and so forth, relating to computer security that have been published in the last decade or so. In any case, few of these have the weight of actual system developments behind them. This section summarizes those projects over the last 10 to 15 years (including currently underway and planned projects) that have included significant efforts to provide secure software systems. Although there have been no deliberate omissions to the list, it is possible that some developments have been overlooked. The following questions were asked about each system:

1. When did its development begin?
2. Who sponsored it (i.e., paid for it)?
3. Who built it?
4. What were its security goals?
5. What approach was used to reach these goals?
6. Were formal specifications used? If so, in what language were they written? Who wrote them?
7. Was any verification done? If so, what tools were used? Who used them?
8. What hardware was the system built on?
9. What programming language(s) were used?
10. If it was built, did the system perform adequately for some practical use?
11. If installed, was it certified for operation with classified data? If so, what mode?
12. Was it/is it installed? Where?
13. What lessons were learned?

The answers to these questions (except the last) are given in Tables 1 to 3. The abbreviations used in these tables are listed in Table 4. Brief comments about each project, summarizing noteworthy aspects and lessons learned, are provided in Appendix A. The reader is cautioned that some of this information is subject to change (particularly for projects underway or planned) and parts of it are the personal conclusions of the author of this report. The bibliography (Appendix B) provides additional sources of information on the projects listed in Tables 1 to 3, but the literature consists largely of technical reports; old reports may be hard to obtain, and reports on new projects have yet to be written.

EXPERIENCE WITH TECHNIQUES AND APPLICATIONS

To distill the lessons from these projects, it is helpful to view them according to the various techniques they have used and the applications that have been tried. Listed alphabetically in this section are various technical concepts and application areas for developing secure systems, together with comments concerning experience with each.

Table 1 — Projects Completed*

System	When Started	Who Paid	Who Built	Goals	Approach	Form Spec.	Verific.	Hardware	Prog. Lang.	Prf./ Cert.	Where Installed
ADEPT-50	late 60s	ARPA	SDC	GP TS w/ security	HWM model objects	no	no	IBM /360	asm, MOL	Y S	NMCSSC, AFCP, CIA, SDC
MULTICS	mid 60s	NSF? ARPA?	MIT	info. utility	VM, rings segments	no	no	GE 645	EPL (PL/I)	Y N	MIT, BTL, etc.
MULTICS Sec. Enhance.	early 70s	AF	MIT/ MTRE	secure MULTICS	Fix code, B+LP mod.	no	no	GE 645	EPL (PL/I)	Y M?	Pentagon AFDSC
MULTICS Kernel	mid 70s	AF?	MIT/ MTRE	kernel for MLTCS	restructure existing impl.	some	no	NA	NA	NB N	spec only
MITRE Brassbd. Kernel	early 70s	AF	MTRE	kernel prototype	B+LP mod. as base	yes	manual	PDP/ 11	SUE-11	D N	MITRE
MITRE Secure UNIX	mid 70s	AF	MTRE	secure UNIX	kernel w/ emulator	yes	no	PDP/ 11	C	D N	MITRE
UCLA DSU	early 70s	ARPA	UCLA	data sec./ UNIX	kernel w/ emulator	yes	some	PDP/ 11	UCLA PSCL	D N	UCLA
MME (SIGMA, Hermes, MIT-DMS)	late 70s	ARPA Navy	ISI BBN MIT	MLS msg. system test	built on pseudo-kernel	no	no	PDP/ 10	BLISS, BCPL, ?	Y S	CINCPAC J-3
SHARE-7	mid 70s	Navy	FCD-SSA	GP TS w/ security	based on kernel post-hoc VMM arch.	no	no	AN/ UYK-7	CMS-2	Y (C)	FCDSSA locs
DAMOS	1979	CR	CR	OS for communications appl.	kernel w/TP's on capability arch.	VDL	no	CR80	?	Y S	?

*For explanation of abbreviations see Table 4.

Table 2 — Projects Under Way*

System	When Started	Who Paid	Who Built	Goals	Approach	Form Spec.	Verific.	Hardware	Prog. Lang.	Prf./ Cert.	Where Installed
KVM/370	1976	ARPA AF DCA	SDC	security retrofit VM/370	kernel w/ VMs, TPs	TLS, SLS, I-J	TLS ITP	IBM/ 370	Jovial J3	NC (M)	(NATC) (DCA)
KSOS (KSOS-11)	late 70s	NSA ARPA Navy	FACC	Secure UNIX	kernel, emulator, TPs	TLS SPCL	TLS SRI	PDP-11	MDLA	N (M)	Logicon, MITRE
SCOMP (KSOS-6)	late 70s	HYWL ARPA DCA, NSA, Navy	HYWL	Secure UNIX	kernel, emulator, TPs, hdwe assist	TLS SPCL HYWL	TLS SRI HYWL	HYWL Lvl 6	UCLA PSCL, C	NC (M)	(MITRE) (NSA)
ACCAT GUARD	late 70s	Navy ARPA	LGCN	sanitize/ filter DBMS to DBMS	TPs on KSOS	some SPCL? GYP	(some)	PDP-11	C, MDLA	NC (M)	NOSC Logicon
LSI GUARD	1980	Navy	IPS	single operator ACCAT GD	TPs on bare hardware	(Euclid)	No	LSI-11	Euclid	NC (M)	?
FORSCOM GUARD	1980	DCA	LGCN	filter— terminal-to-host, w/operator	TPs on UNIX	(GYP)	(GYP)	PDP-11	C (MDLA)	D (M)	Army FORSCOM
SDC Comm. Kernel	late 70s	?	SDC	secure comm. proc.	tailor UCLA kernel	NO	NO	PDP-11	UCLA PSCL	? ?	SDC
AUTODIN II	late 70s	DCA	WU, CSC, FACC	secure comm. proc.	kernel-based arch.	Yes, post hoc	Yes ITP	PDP-11	C, asm	NC (M?)	(DIN I sites)
SACDIN	late 70s	AF	ITT IBM	secure comm. proc.	kernel-based arch.	TLS SPCL	TLS SRI SYTK	IBM Series I	asm	NC (M?)	(SAC sites)
COS/NFE	late 70s	DCA	DTI	secure WWMCCS/ DIN II NFE	"HUB" TM kernel, trusted modules	TLS I-J	TLS ITP	PDP-11	PSCL	NC (M)	(WWMCCS sites)
GNOSIS	early 70s	TymS	TymS	segregate client data	Capabilities	No	No	IBM/ 370	asm	NC (M?)	Tymshare
PSOS	1980	NSA	FACC HYWL	Secure Op. Sys.	Spec. and Ver. entire OS.	Yes	Yes (MAN)	HYWL	—	NB (M)	—
Message Flow Modulator	1981	Navy	UTX	filter & transform output	TPs, full ver.	GYP	GYP	LSI-11	GYP (M)	NC	(OSIS)
SASS	1978	Navy	NPGS	secure archival file sys.	multi-micro-kernel	no	no	Zilog 8000s	asm?	NC (M)	PGS

*For explanation of abbreviations see Table 4.

Table 3 — Projects Planned*

System	When Started	Who Pays	Who Builds	Goals	Approach	Form Spec.	Verific.	Hardware	Prog. Lang.	Prf./Cert.	Where Installed
RAP GUARD	early 80s	NASA	MTRE	filter-terminal-to-host, no oprtr.	TPs on KSOS ?	?	?	Z80?	?	N ?	Space Shuttle planning
KAIS GUARD	early 80s	KAF?	MTRE	" ?	" ?	?	?	Z80	?	NC ?	Korea?
WWMCCS Local Net	early 80s	DCA	–	MLS nets for WWMCCS nodes	Sep. nets w/GUARD for internet	?	?	?	?	NC (M)	(WWMCCS sites)
MITRE MLS terminal	1980	AF?	MTRE	secure intelligent terminal	process per processor, MLOs	No	No	?	?	NC (M?)	(MITRE)
NRL MMS/TDMS	1981	Navy	NRL SDC BBN	MLS msg. system/ database	appl.-based sec. model, prototype	?	?	?	?	D (M)	–
MITRE DBMS Kernel	1981	Navy	MTRE	secure MRS DBMS	tailored kernel	(Y) TLS	(Y?)	SCMP	C?	NC (M)	(MITRE)
EIFEL 2	late 70s	GDR?	GDR SMNS TLF?	MLS air-traff. ctl.	kernel based?	?	?	?	?	? ?	–
Canadian Maritime Command	early 80s	?	?		?	?				?	?

*For explanation of abbreviations see Table 4.

Table 4 — Abbreviations Used in Tables 1 to 3

? — data unknown or uncertain
( ) — enclosed data indicate plans, not accomplishments

Performance Codes

N — No — performance not adequate for operational use
Y — Yes — system could be used operationally
D — Demo — system built as prototype or demo only
NC — System not yet complete enough for evaluation
NB — System never built

Certification Codes

M — Multilevel
C — Controlled
S — System high
D — Dedicated
N — Not certified

Hardware

GE 645	Specially modified GE 635. When Honeywell bought GE, the numbers changed. This is the original MULTICS machine number. New Honeywell models have since been introduced and the commercial version of MULTICS runs on the newer hardware (Honeywell 6180).
Lvl 6	Honeywell Level 6 minicomputer
SCMP	SCOMP — Honeywell Level 6 with added Security Protection Module (SPM)

Companies/Laboratories

BBN	Bolt Beranek and Newman, Inc.
CR	Christian Rovsing (Denmark)
BTL	Bell Telephone Laboratories
CSC	Computer Sciences Corp.
DTI	Digital Technology, Inc.
FACC	Ford Aerospace and Communications Corp.
HYWL	Honeywell
IPS	I.P. Sharp Associates
ISI	Information Sciences Institute (USC)
LGCN	Logicon
MTRE	MITRE Corp.
SDC	System Development Corporation
SMNS	Siemens (W. Germany)
SRI	SRI International
SYTK	Sytek
TLF	Telefunken (W. Germany)
TymS	Tymshare, Inc.
WU	Western Union

(Table continues)

Table 4 (Continued) — Abbreviations Used in Tables 1 to 3

Government Organizations

AFCP	Air Force Command Post (located in Pentagon)
ARPA	DoD Advanced Research Projects Agency
AFDSC	Air Force Data Services Center
AF	Air Force
CIA	Central Intelligence Agency
CINCPAC	Commander-in-Chief, Pacific
DCA	Defense Communications Agency
DIA	Defense Intelligence Agency
FCDSSA	Fleet Combat Direction Systems Support Activity
NATC	Naval Air Test Center, Patuxent River
NMCSSC	National Military Command System Security Center
NPGS	Naval Postgraduate School
NRL	Naval Research Laboratory
NSA	National Security Agency

Languages/Tools

asm	Assembly language (for whatever machine indicated)
GYP	Gypsy — programming and assertion language, with verification tools, developed at University of Texas
I-J	Ina Jo™ — specification language supported by SDC
ITP	Interactive Theorem Prover — supported by SDC, used w/Ina Jo
MAN	Manual proofs
MOL	Macro Language for IBM/360
MDLA	Modula
PSCL	Pascal
SPCL	SPECIAL — specification language, tools available from SRI
UCLA PSCL	Pascal-to-C translator implemented at UCLA for a "verifiable" Pascal subset
UNIX	UNIX™ operating system, originally developed by Bell Laboratories
VDL	Vienna Definition Language

Miscellaneous

AM	Access matrix
B+LP	Bell and LaPadula
GD	GUARD
GP	General purpose
HWM	High water mark
MLO	Multilevel object
MLS	Multilevel secure
SLS	Second-level specification
TLS	Top-level specification
TP	Trusted process
TS	Time sharing

Capabilities

A capability is usually defined as an unforgeable ticket that grants its holder a specific kind of access to a particular object; capabilities can be implemented as virtual addresses, with additional bits to define allowed access modes. Capabilities provide a mechanism for controlling access to objects, not for implementing security policy. The provision of security depends on the design of the system, which may be based on the use of capabilities. Nevertheless, they are an appealing tool for structuring the implementation if the appropriate hardware is available. Without the appropriate hardware, there is considerable evidence that capability-based systems perform poorly (but see GNOSIS). Architectures based on the manipulation of capabilities are still more touted than tried, although hardware architecture trends seem to be moving in this direction (e.g., the Plessey S250, the IBM System 38, and the Intel 432). The UCLA DSU kernel (and hence the SDC Communications Kernel) employs the concept of a capability but is constrained to use the addressing hardware of a PDP-11. PSOS is intended to employ a capability architecture and will be an interesting test case, if it is built. The GNOSIS effort claims to be based on an implementation of capabilities on standard IBM 370 hardware. No secure system developments using the Plessey hardware have been reported.

Databases

None of the secure system development efforts so far have tried to deal with the complexities of securing a database management system shared among a variety of users. There is substantial theoretical evidence that statistical databases are impossible to secure against inference, but there are some techniques that can make it more difficult for information to be compromised.

Encryption

There are increasing efforts to apply encryption within computer systems. Protection and distribution of keys then becomes a critical issue. Because they were generally trying to secure the operating system itself, which would presumably control access to keys, none of the systems listed in Tables 1 to 3 used encryption as a technique for protecting information.

Kernels

Many of the projects listed above sought to demonstrate the practicality of the security kernel approach. A security kernel is a small subset of a system that is responsible for its security and so must provide *complete* validation of program references, must be *isolated* (tamperproof), and must operate *correctly* according to a stated security policy. The results so far are mixed, at best. The three-layer approach (user programs running on an operating system emulator running on a security kernel) has yet to produce a system efficient enough for operational use. The KSOS represents the most serious attempt to use this approach to build an operational system, and it has fallen far short of its performance goal of approximately 50% degradation of unmodified UNIX on the same hardware. Some observers lay the blame at the feet of the PDP-11 hardware. Performance data are just becoming available on the kernel for the SCOMP, which is built on hardware specifically modified to assist security. Honeywell now plans to build a minimal operating system interface, including a simple file system, outside of the kernel, instead of developing a full emulator. The KVM/370, which is more on the order of a retrofit kernel, has been brought up on IBM and Amdahl hardware, and early indications are that it offers about 50% of the performance of unmodified VM370 on the same hardware. SHARE-7 is now described as having a security kernel, but it appears to have been constructed more on the virtual machine model, with kernel terminology superimposed on it after the fact. The SDC Communications Kernel, which seems to be the kernel-based system closest to delivering a useful product at present, is really a two-layer approach: the kernel is carefully tailored for the application and the application runs directly on the kernel. The AUTODIN II kernel is rarely cited as a good example of anything. Recent reports from the Christian Rövsing company in Denmark may indicate a significant advance in the development of an operational system based on a security kernel.

Measures

There are no useful quantitative measures at present for defining the relative "security" of various systems. None of the projects listed in the tables has addressed this problem seriously. The only realistic measures would seem to be the difficulty of penetration or the rate of unauthorized information flow out of the system under specified conditions.

Network Architectures

Although papers are being published in this area, novel architectures for secure networks are lacking. While some of the efforts described have developed machines for use as network switches (AUTODIN II, SACDIN), none has provided innovative solutions to network security problems. Topics under study include the use of public key encryption algorithms and making communication protocols secure. End-to-end encryption does appear to be coming closer to being practical. The WWMCCS local network project may contribute something to the solution of this problem. (See also Protocols.)

Penetration Studies

Every serious attempt that we know of to penetrate a particular system has succeeded. Information on penetration studies applied to the systems listed in the tables is sparse. A penetration study of VM/370 (CP/67) done by SDC did conclude that it was significantly more difficult to break into that system, which is based on virtual machines, than into some others.

Protocols

See Network Architectures. The major problem is how to limit the unauthorized information flow possible in the control information, which often has to be transmitted in the clear (unencrypted).

Risk Assessment

A risk assessment attempts to characterize, within a specific installation, the assets of a system, the threats to them, and the system's vulnerabilities to those threats. Risk assessments of computer installations are being conducted at a number of installations. Since risk assessments consider conditions at a specific site, the most general assessment that can be done by the developer is an analysis of the vulnerabilities of the system in general. Vulnerability analyses may be classified, since they may reveal exploitable system flaws. There has been an example risk assessment conducted on GUARD, partly to evaluate risk assessment methodology. Of the completed systems, AFDSC MULTICS and ADEPT-50 are the most likely to have had such analyses done. A risk assessment of SHARE-7 has been conducted. Presumably, a vulnerability analysis will be done for AUTODIN II. The most difficult problem in any risk assessment is what value to attach to the data at risk.

Security Models

Of the projects listed in the tables, the MITRE kernel, MULTICS security enhancements and kernel, KSOS, SCOMP, GUARD, MME message systems, AUTODIN II, KVM/370, and SACDIN, all are based on the Bell and LaPadula model. The UCLA kernel implemented underlying controls for capabilities, with the specific Bell and LaPadula rules implemented in a "policy manager" module. This uniformity seems to be caused more by government direction than by superiority of the model. In fact, it is now being recognized that models that can more closely reflect applications are necessary, particularly in dealing with data bases. Still, the Bell and LaPadula model (and its restatement in terms of information flow) is the baseline for the field (see Ref. 2 for a detailed discussion).

Specification Techniques

Parts of several of these systems have been specified formally. Systems and tools employed include SPECIAL/HDM/Boyer-Moore Theorem Prover (SRI), Gypsy (University of Texas), Ina Jo/ITP (SDC), and AFFIRM (ISI). There is evidence (e.g., KSOS and SCOMP) that programmers can write formal specifications (after some training) and that some errors (security flaws) are exposed by the use of automated tools to check (verify) the specification. However, the major benefits for security seem to be that other humans (system developers or reviewers) can understand what the specification says and find flaws (security or other) by reviewing the specification manually, despite the often forbidding appearance of a formal specification. This finding is partly a comment on the state of the tools available for formal specification and verification. Evidence from these projects is that the tools currently available, though improving, are best characterized as research vehicles, not production-quality aids to software development.

Verification Techniques

Most of the comments on specification techniques apply here as well. Verification of security properties of top-level specifications seems to be within (but fairly near the edge of) the current state of the art. Verification of actual code (in some compilable language) is still very hard for the available tools and usually requires considerable human intervention. Despite apparent progress in research in this area, the present systems are some distance from practical use as system development tools. The benefits to be had from attempting verification seem to be software-engineering ones: it forces the designer/implementor to think hard about what he is doing and leaves an extensive documentation trail for others to review.

Virtual Machines

The KVM/370 is the prime example of a secure system based on this approach; SHARE-7 is another example. This organization helps in isolating individual users at separate security levels, and in systems where isolation is all that is required, it is the best available approach. However, experience indicates that such isolation is rarely what is really needed; if extensive communication between users and across security levels is required, this organization can get in the way.

Virtual Memory

This is one tool for organizing computer systems that it would seem foolish for a developer of a new secure system to forgo. A persistent problem with many of the efforts listed in the tables has been the small virtual address space of the PDP-11 and its approach toward virtualization of devices.

ADVICE FOR THE DEVELOPER

What are the lessons for the developer of a new system? It is important to distinguish between technologies that are available and useful today and research approaches that appear promising but are unproven. There is, at present, no proven technology that can assure that the system being developed will be secure. Those techniques that have been tried have met with varying degrees of success. It is difficult to give objective measures for these degrees of success because there are no good measures that can be applied to rank the security of various systems.

Listed below are the best available approaches for incorporating security into a system under the stated constraints. Each is listed under the appropriate phase of the system development cycle. Many of these simply represent good system design and software engineering practices.

Requirements

Determining the requirements for security in a system is crucial, because they will affect the entire structure of the system software. Security is not an "add-on" feature. It must be incorporated throughout a system, and the statement of security requirements for a system should reflect this fact.

There are at present four different modes of operation for which systems processing classified information can be accredited:*

Dedicated: All system equipment is exclusively used by that system, and all users are cleared for and have a need to know for all information processed by the system;

System high: All equipment is protected in accordance with requirements for the most classified information processed by the system, and all users are cleared to that level, but some users may not have a need to know for some of the information;

Controlled: Some users have neither a security clearance for nor a need to know for some information processed by the system, but separation of users and classified material is not essentially under operating system control;† and

Multilevel: Some users have neither a security clearance for nor a need to know for some information processed by the system, and separation of personnel and material is accomplished by the operating system and associated system software.

Definitions of these modes are provided in DoD Directive 5200.28 [3].

A Request for Proposal (RFP) should state what modes of operation are needed for the system initially and whether future operation in other modes is planned. However, if the requirement for security is isolated and stated baldly ("The system shall be secure"), bidders may view the security requirement separately from other system requirements and so may propose infeasible solutions.

The system architect should consider the system's security requirements as part of its functional requirements from the start. In this way intelligent trade-offs can be made where required and a coherent design, integrating the needs for functionality and security, can be obtained. If this procedure is not followed, bidders may claim that they can build the system only to discover that requirements conflict when they are well into the development.

One illuminating example is that of the military database system that normally contains only unclassified data, but during crises some of its contents may be classified. Because the requirement to handle classified data was not implemented initially, its users must either finance a duplicate system, attempt to retrofit security to the existing system, or operate in a manual mode during crises.

One way the system architect can integrate the security requirements with the functional requirements explicitly is to specify the flows of information (especially classified information) that will occur in the system and the flows of authorization. Particular places to look for problems are in the user interface, in any operations that cause information to flow into or out of the system, and any places where the classification of information could be changed.

*Additional constraints may be placed on systems processing compartmented information.

†"Essentially" is not further defined in the official documents. In practice, controlled mode seems to be applied to systems that would be considered as operating in a multilevel mode, but which bar users with clearances below some specified level from having access to the system.

Trade-offs in the provision of security should be identified and assessed as early as possible. For example, physical controls and computer hardware and software controls are alternative techniques for protecting information stored on computers. If they are not assessed as alternatives early in the system development, however, physical security will be the *de facto* choice, because it is much easier to provide after the fact than hardware and software controls. Unfortunately, physical controls frequently restrict functionality more than comparable software controls.

Within the software and hardware design there will be trade-offs as well. Many of these only need to be addressed in the design, but the system requirements should provide guidelines on such matters as the granularity of protection needed, critical pieces of information that may require special protection, acceptable bandwidths for leakage channels, and so on. The designers will be forced to make decisions on these questions whether or not guidelines are provided; it is in everyone's interest that these decisions be informed, not *ad hoc*.

Design

For development of a multilevel secure system, security must be considered early and often during the design phase. As with requirements, it cannot be added to an existing design. The most prominent design strategy at present is the security kernel approach, but it is not a proven technology. There is evidence that use of a security kernel can impose intolerable functionality or performance burdens. Improved hardware may ease these burdens somewhat, but there are still substantial questions about the viability of this approach. At present, a system designer would be well advised to do the following:

- Study the functions of the system, focusing on requirements for the flow of classified information and the interface with the user. Specifically, note under what conditions sensitive information is disclosed or modified, has its classification changed, or enters or leaves the system. Include mechanisms to audit the use of system functions that may leak information.

- Construct a simple model of the flow of information and authority within the system. The model need not be formal, but should be brief, precise, and simple enough to be understood by both the implementors and the users of the system.

- Keep the model and the design consistent. If changes are required to either, make corresponding changes to both.

- Develop a hierarchical set of design specifications. Include a top-level specification that reflects the basic functions of the system and the information-flow model, and a program specification that is sufficiently detailed to allow outside reviewers (and new personnel) to review the code structure and to assess the information flows and authorization mechanisms it will have. As the design is created, have it reviewed at regular intervals both by the potential users and by individuals knowledgeable in computer security. KSOS, SCOMP, the SDC Communications Kernel, and other projects have used this approach with good results. Use of formal specifications may be helpful, but their use should be contingent on training of the implementors to the extent that they are competent to read and update them. The software tools presently available for formal specification and verification are still primarily research vehicles, although this situation may change within a few years.

- Choose hardware that reduces security problems. Generally, hardware that provides good mechanisms for isolating different computations, simple and efficient ways to control the flow of information between isolated contexts, and a uniform way of treating different kinds of objects is desirable. The following features will help in the implementation of secure systems: virtual memory, with controlled access to the mapping registers; a device interface that offers the possi-

bility of a uniform treatment of memory, files, and devices; and the ability to change addressing contexts rapidly. Several machine states that restrict access to critical portions of the instruction set are not required if all accesses to data are mediated by the virtual memory mechanism, but they can be used as another way of protecting critical operating system data (e.g., the contents of the mapping registers).

Implementation

The implementation should employ a language for which there is a well-understood, reliable compiler. There are some languages (e.g., Gypsy, Euclid) that have been designed with the intention that programs written in them be verified and others (Pascal, Ada) for which "verifiable subsets" have been proposed. Experience to date indicates that a reliable compiler and the disciplined use of a conventional language are preferable to a relatively untested compiler and a "verifiable" language. Assembly language should be avoided whenever possible.

If it becomes necessary for the code to deviate from the specifications, the specifications should be updated in parallel with the code changes. Good coding practices benefit the security of the system as well as its reliability and maintainability. Careful attention should be paid to configuration control.

Verification and Testing

Automated verification that a given formal top-level specification obeys the Bell and LaPadula security model is within the state of the art, but it is far from routine. Experience shows also that virtually all applications require functions that violate the Bell and LaPadula model. Thus, if the specification is verified to enforce this model, the implementation will deviate from it in some respect. At present, automated verification of the security properties of substantial amounts of code is beyond the state of the art. These techniques hold promise for the future, but there is substantial risk in employing them in a system development that starts next week. The verification of specific properties of a specification or small pieces of code can be done by hand or, if written in suitable languages, with machine assistance. Currently, the principal benefit of doing so is that the exercise focuses a good deal of attention on the specification or code at hand and requires the doer to think the problem through very carefully. A few, but not many, security flaws in systems have been found through use of verification techniques.

Thorough testing continues to be a necessary partner to careful design and implementation of systems to be operated in a multilevel secure mode. When test plans are constructed, specific attention must be given to testing the security provisions of the system. If possible, the developer should arrange for penetration tests and estimate the bandwidths of leakage channels that cannot be eliminated.

Operation

From the standpoint of security, the important aspects of system operation are the controls over changes to the system software and configuration and the conscientious use of the security mechanisms provided by the system. Configuration control is of central importance in the operation of a multilevel secure system, and the design of the system itself should assist in this task. The maintenance of various levels of specification, good coding practices, and use of high-level languages all help.

One often neglected aspect of operations is the monitoring of the audit trails that are routinely collected: usually they are too voluminous (and boring) for us to expect a person to do a reasonable job of checking them. Automated tools for this purpose need close attention, since defeating the tools becomes equivalent to defeating the auditing controls.

SUMMARY

There is no philosopher's stone to turn a given system (or system design) into a multilevel secure version of the same system. The advice given above boils down to:

- Consider security requirements in conjunction with the functional requirements of the system.
- Think about security throughout the design and implementation of the system.
- Use the best available software-engineering technology.
- Be skeptical. Many "modern" ideas do not yet work.

ACKNOWLEDGMENTS

I thank Dave Parnas for asking the question that inspired this report. The information on which the report is based came from individuals too numerous to mention, but in whose debt I remain. Dave Parnas and Connie Heitmeyer provided reviews of earlier drafts of the paper that led to significant improvements in it, and H. O. Lubbes of NAVELEX provided encouragement and support. The responsibility for all opinions (and any remaining errors) in the paper is mine.

REFERENCES

1. *Software Engineering Principles*, Notebook for NRL Software Engineering Course, 1980, available as NTIS AD-A087-997.

2. C.E. Landwehr, "Formal Models for Computer Security," ACM Computing Surveys **13** (3), 247-278 (Sept. 1981).

3. DoD Directive 5200.28 of 18 Dec 1972, first amendment, change 2, 29 April 1978

Appendix A
COMMENTS ON SECURE-SYSTEM DEVELOPMENTS

PROJECTS COMPLETED

ADEPT-50: This system was the first to be based on a formal model of security. This model (called the High Water Mark model) allowed write-downs (with authorization) but restricted read-ups. Installed in the Pentagon (AFCP, NMCSSC), CIA, and SDC, it ran for several years. It was certified for system-high operation only.

MULTICS: The MULTICS operating system (and the MULTICS hardware) attempted to pay a good deal of attention to protection, if not to security. The most significant innovation in this respect was probably the concept of rings of protection, in which inner (lower numbered) rings are more privileged than outer rings. The architecture generalized the concept of a two-state (supervisor and user) machine to that of an n-state machine, with one state for each ring. For the original MULTICS machine, $n = 16$. In practice, nearly all of the original MULTICS privileged software was in the innermost ring. This result was probably partly due to the fact that handling ring crossings was initially done in software, making it expensive (time consuming). Later, hardware for ring crossing was added and there were efforts to distribute the operating system components across several rings. These efforts were not very successful. MULTICS also included an implementation of access lists for files (segments) and a hierarchical file system. Verification and security models were not considerations in the MULTICS development. Performance of MULTICS was poor for several years, but gradually improved to the point that Honeywell now markets MULTICS as a supported product. The current hardware is the Honeywell 6180.

AFDSC MULTICS (MULTICS Security Enhancements): The Bell and LaPadula model was applied to MULTICS — MULTICS objects had classifications attached to them and the *-property and simple security condition were enforced. No effort was made to isolate the security-relevant code or to restructure MULTICS into a kernel. The system was installed in 1974 at the Air Force Data Services Center in the Pentagon and has been certified to operate with both TS and S data and with some users cleared only to S.

MULTICS Security Kernel: Following the MULTICS security enhancements, this was an effort to see if the MULTICS operating system could be restructured as a security kernel and supporting software. Studies were done by Honeywell and MITRE separately. At the same time, MIT studied the MULTICS supervisor to see how much of it could be moved out of Ring 0. The MIT study indicated considerable reduction of Ring 0 code was possible without complete restructuring of the system as a kernel. Also, MIT identified potential performance problems with kernel. This kernel was never built.

MITRE Brassboard Kernel: This was a prototype security kernel developed for a PDP-11. Kernel operations were based on the models constructed by Bell and LaPadula. Performance was poor, but good performance had not been an objective of the project. Both top-level and low-level specifications were written, and extensive manual verification of the kernel operations and the correspondence between specification levels was performed.

MITRE Secure UNIX: When MITRE tried to put a UNIX emulator on top of the brassboard kernel, they found that the match was so poor that a new kernel was needed that provided functionality

more suited to UNIX. Formal specifications were written (referred to as "Parnas" specifications), but no verification was done.

UCLA Data Secure UNIX: This was another prototype security kernel for a PDP-11 (parallel to MITRE's), but it was carried much further than the MITRE implementation effort. Its architecture was based on an implementation of "capabilities" for the PDP-11. A prototype was developed and fitted with a UNIX user interface, but performance was very slow. A separate security model was developed for this prototype, based on "data security"; the Bell and LaPadula model could be enforced by a policy manager module running outside the kernel. A strong effort to keep the kernel small resulted in a fair amount of code outside the kernel that did have some security-relevant functions (e.g., the policy manager). Hence, comparing code sizes of the UCLA kernel and others must be done carefully. Much effort was expended on formal verification, in cooperation with ISI and the XIVUS theorem prover. The effort seems to have been hampered initially by the lack of a high-level specification of the system. This lack seems to stem from the philosophy that the only verification that counts is verification of the lowest level assembly code. Ultimately, the verification of about 35 to 40% of the kernel code was claimed as a feasibility demonstration that the entire kernel could be verified, given sufficient resources.

Message Systems for Military Message Experiment (MME): Three message systems were developed in competition: Hermes, by BBN; SIGMA, by ISI; and MIT-DMS, by MIT. The ISI system won the competition and was used in the MME. SIGMA was designed as though it were running on a security kernel, although the system underneath it was not in fact a kernel-based system. A "trusted job" was introduced to allow required operations that violated the Bell and LaPadula model. SIGMA required users to confirm activity that could cause insecure information flows. In practice, most users confirmed every operation requested without understanding or thinking about the implications.

SHARE-7: This Navy system developed at FCDSSA, and implemented on the AN/UYK-7 computer, is in operational use at several FCDSSA installations. The system was originally designed as a virtual machine architecture, to provide multiple AN/UYK-7's to users. More recently, security-kernel terminology has been used to describe its security structure. It was written in CMS-2. It is now undergoing security certification procedures for operation in the controlled mode. (Originally, certification for multilevel mode had been sought.) It is one of the first systems the Navy has nominated for evaluation by NSA's Computer Security Evaluation Center.

DAMOS: This is an operating system developed by the Danish company Christian Rovsing (CR) to run on its own CR80 computer. The computer and the operating system are described as capability-based, and the operating system includes implementations of a security kernel and trusted processes. These concepts seem to have been developed by CR independent of US efforts in security kernels. DAMOS is a successor to an earlier system, AMOS, and is intended to provide fault tolerant operation in communication-system applications. Systems in which CR80s are or will be used include NICS TARE, a NATO system for automating paper-tape relay operations; FIKS, a Danish DOD system for message, packet, and circuit switching; and CAMPS, a NATO-SHAPE message system for communication centers.

PROJECTS UNDER WAY

KVM/370: This is an SDC project to install a kernel underneath the IBM VM/370 operating system (a virtual machine-based system). Presumably, the system is near delivery. It has been brought up on IBM and Amdahl hardware. Performance is claimed to be acceptable, though significantly slower (approximately by half) than standard VM. This seems to be the only

current project dealing with the complexities of a large-scale time-sharing system (though some argue that this is really a small system running on large hardware). If successful, it will primarily provide multiple virtual machines at different security levels, with restricted communication between machines handled via shared "minidisks."

KSOS (also known as KSOS-11): This represents an attempt to build a commercial prototype kernel-based system on a PDP-11/70. An emulator on top of the kernel was to provide a UNIX interface to users. Full verification of the security properties of the kernel top-level specification (TLS) (written in SPECIAL) and demonstration proofs of correspondence of code (written in MODULA) to specifications were planned. The contract with FACC was terminated in December 1980. Performance with the UNIX emulator and user software layers was poor. Verification of the kernel TLS (using the Boyer-Moore theorem prover at SRI) seems to have been successful. The code proof demonstrations were done by hand. The kernel by itself may prove useful as a base for applications, but the emulator ended up duplicating kernel functions and performed poorly in combination with it. The Navy has funded Logicon to look into building the GUARD application directly on the kernel.

SCOMP (Also known as KSOS-6): This project was intended to parallel KSOS-11 but uses Honeywell Level 6 hardware. The U.S. government funded development of a hardware box called the Security Protection Module (SPM) and the kernel software specification and development. The SPM monitors transfers on the bus without CPU interference, providing faster mediation and enhanced virtual memory/capability structure. The kernel and hardware are nearly complete; kernel performance on the hardware is being tested now. Original plans called for the development (funded by Honeywell) of a UNIX emulator, as in KSOS-11; it now appears that only a minimal operating system interface, containing a simple file system, will be built. Specification of the kernel was done in SPECIAL, with verification using SRI tools. The verification was substantially completed, but a few modules proved too large to be passed through the SRI tools. No code proofs are planned. The kernel is coded in UCLA Pascal, compiled via Pascal-to-C (UCLA). The C-to-Level-6 compiler was built by DTI. Trusted processes are to be specified in Gypsy, but the current contract does not cover verification of them.

GUARD (ACCAT, LSI): ACCAT GUARD (the original version of GUARD) provides a "trusted" path between two database systems or networks that operate at two different security levels. It supports operators who monitor requests made from the low database to the high database and sanitize the responses returned. The trusted process that performs downgrading was specified in Gypsy. GUARD was initially planned to run on top of the KSOS UNIX emulator, but it is now being reevaluated to see if it can run directly on the KSOS kernel. A prototype version was developed to run on an unmodified UNIX. LSI GUARD is an attempt to implement the ACCAT GUARD functions directly on the hardware of a DEC LSI-11 (without a kernel) in Euclid. It allows only a single operator, while ACCAT GUARD can have multiple operators.

GUARD (FORSCOM, RAP, KAIS): Each of these GUARDs is designed to act as a filter between a terminal and a host computer rather than as a query monitor between two databases. FORSCOM GUARD was built by Logicon based on modifications to ACCAT GUARD, and it is being tested by the Army to filter traffic between several WWMCCS terminals and a single WWMCCS host. The system requires a single human operator, who screens traffic for all of the terminals. The FORSCOM GUARD programs (unlike those of ACCAT and LSI GUARD) include substantial information about the semantics of the likely user activities on the WWMCCS host to enable more accurate filtering. FORSCOM GUARD was built on top of standard UNIX because KSOS was not available. Performance of the system was adequate for the Army to use it in a test mode, but it brought some user complaints.

SDC Communications Kernel: This is a project to modify the UCLA prototype PDP-11 kernel for use in communications applications (packet switching, etc.). The code is written in UCLA-Pascal and compiled using the UCLA Pascal-to-C translator. The kernel was heavily modified and tailored to the application; no operating system emulator was used. Performance seems satisfactory at this point, but some performance tests are still pending. Pascal-to-C is not recommended for further use; it takes about 5 minutes of unloaded PDP-11/70 time to compile each of 60 submodules, or 5 hours for the entire system. The requirement for verification of the specification was dropped early in the project; a requirement for "verifiable" code motivated the choice of Pascal-to-C, but the only verification actually done has been manual examination of the specification for information channels.

AUTODIN II: This is a packet-switched communication network for military use, provided by DCA. Western Union is the prime contractor, with Ford Aerospace (a different group from the KSOS group) and Computer Sciences Corporation as software subcontractors. The AUTODIN II RFP called for a kernel-based system for packet switching, but without adequately defining what a kernel is or what the requirements for formal specification and verification were. There have been many problems in its development, including a court fight over the definition of "formal specification." Eventually, FACC wrote the code and [redacted], and to some extent verified (using Ina Jo and ITP), the specification after the fact. The system has apparently passed its security and system tests, but the date of its availability to users is still uncertain.

SACDIN: Originally named SATIN IV, this was to be a packet switching net for SAC, but the Air Force was directed to use AUTODIN II for the network backbone. The SATIN IV RFP was thought to be better (with respect to security) than AUTODIN II; the SACDIN development is somewhat less ambitious than the original plan. IBM is working as a subcontractor to ITT, building a kernelized system. A top-level specification was written and verified, but the implementation is in assembly language (for IBM Series 1 computers) and no code proofs are planned. No mapping between the specification and the code has been constructed.

COS/WFE: This DCA project with DTI is an effort to build a secure network front end for WWMCCS using the DTI-proprietary HUB architecture (HUB a trademark of DTI). SDC is a subcontractor to DTI for specification and verification, using Ina Jo and ITP. According to DTI, the HUB executive is a security kernel. Top-level and second-level specifications for the HUB have been written, and security criteria for it have been developed and proven to hold for the specifications (both levels). The security criteria are based on maintaining a strict separation of "security level sets." The HUB structure includes many modules that will process data at several security levels; these are termed "trusted" modules. Whether formal low-level specifications will be written and verified against the code is still an open question. The whole system is intended to be small, fast, and application-driven. It is being written in Pascal for PDP-11 architecture. An early specification was written in Euclid.

GNOSIS: This is a new operating system built by Tymshare, claimed to be based on an implementation of capabilities for IBM 370 architecture. No formal specification or verification was done, but an important motive in developing the system was to be able to protect timesharing customers' data from theft, so much attention was paid to protection (though not to military security). The system is in operational use by its developers, but is not commercially available as yet.

PSOS: An initial specification for a "Provably Secure Operating System" (PSOS) was written by SRI in the mid to late 1970s. This document was used as part of a product description for a two-phase contract finally awarded in early 1980 to Ford Aerospace as prime contractor for the software, with Honeywell as a subcontractor, primarily to provide software, but also to provide some help with verification. The goal is a moderate- to large-scale, general purpose, verified (or verifiable) computer system, not kernelized. The first phase is for a design only; the second

phase, if awarded, will be for development. The plan is to build a capability-based system, with proofs of the entire OS specification. It is unclear whether code proofs will be done. Honeywell bid 32-bit minicomputer hardware that looks like a next-generation Level 6, with SPM (see SCOMP). Unsuccessful bidders were Burroughs (software and hardware) and SDC/Univac. The phase-one contract was terminated in May, 1981; the fate of the second phase (for construction) is unknown.

Message Flow Modulator: A flow modulator is a system that receives messages from a source, filters out certain messages, transforms messages that pass the filter, and transmits the resulting messages to a sink. A flow modulator with a null transform is like a GUARD, and a flow modulator with a null filter and an encryption algorithm for a transform is like a standard encryption device. Don Good's group at the University of Texas has developed a simple flow modulator using Gypsy and the Gypsy tools, and has proven some properties about the system. Present work on the system is aimed at applying it to a Navy system.

Secure Archival Storage System: The goal of this project is to apply security kernel technology to a network of microprocessors that store multilevel files. Ultimately, SASS would provide a shared, secure archival storage for a variety of host computers. The initial implementation is on a single Z8000 processor and is being carried out by Roger Schell, Lyle Cox, and a number of graduate students at the Naval Postgraduate School. Although there are no plans to verify the design or the implementation, and the design seems not to have been specified formally, the authors consider the security kernel to be "verifiable."

PROJECTS PLANNED

GUARD (RAP, KAIS): RAP GUARD is to be used by NASA in a shuttle planning system between the terminals and other network components. MITRE is working on the design of this system and a similar one for the Korean Air Intelligence System (KAIS). In each of these three systems, the GUARD component includes some knowledge about the activities the user (at the terminal) is performing on the host, as in FORSCOM GUARD. (e.g.: Was the last command an editing command that has a stereotyped response or a database query that could provide significant information to the user?) They are intended to operate between a single terminal and a single host, and are not intended to require an operator. They are intended to filter out classified material of inappropriate levels without human intervention.

WWMCCS Local Network: The evolutionary path of WWMCCS and WIN is intended to lead to a set of local networks, interconnected by packet switching (AUTODIN II) trunks. (All of this is called the WWMCCS Information System [WIS] architecture.) SDC presented three scenarios for a trusted/multilevel secure local network to a committee convened by DCA in February 1981. The goal was to identify which scenario would be realistic to specify in an RFP to appear in FY82/83. The most conservative scenario presented, which involved separate local networks at different security levels, connected by a GUARD-like security filter box, was recommended. Additional studies of the architecture are to be conducted.

MITRE MLS Terminal: This project is investigating the practicality of dealing with multilevel objects by assigning a separate microprocessor to handle each security level, with a "trusted feedback monitor" to coordinate their activities. A design has been produced based on a word-processing application, and there are plans to implement it.

MMS/TDMS: This is an NRL project (funded by NAVELEX) to design a prototype Military Message System viewed as a Trusted Data Base Management System. A contract was let to SDC to develop design specifications and to work on the proposed MMS security model. BBN has also participated.

MITRE DBMS Kernel: This is a MITRE project to specify and implement a kernel designed specifically to support a database. The SCOMP (Honeywell Level 6 with Security Protection Module [SPM]) is to be the hardware base, and the MRS database management system is the most likely to be used.

NON-U.S. WORK

Germany: EIFEL is a German air traffic control network (perhaps also command and control) that is due for a new generation of hardware and software (EIFEL 2). The German equivalent of MITRE was (in 1980) trying to write a specification for the system that included multilevel secure operation. Siemens, Telefunken, and others were also investigating the problem as prospective bidders.

Canada: The Canadian Maritime Command is apparently attempting to procure a system that may have a multilevel security requirement.

England: There is some interest in the security problem at the Royal Signals and Radar Establishment (RSRE), Malvern. At last report their central interest was control flow analysis of programs and low-level code verification.

Denmark: See DAMOS (p. 16)

Appendix B
BIBLIOGRAPHY

The following list is intended to give the interested reader pointers to further information about each of the systems listed in this report. It is not intended to be a complete list of references on these systems. Although we have tried to include references that are generally obtainable, several of these projects are documented only in technical reports whose availability cannot be guaranteed.

At this writing, the single best reference for current work promises to be the *Proceedings of the Fourth Seminar on the DOD Computer Security Initiative Program*, held at the National Bureau of Standards in August 1981. Some of the "papers" in these proceedings, however, are only copies of the viewgraphs used in oral presentations. These proceedings should be available from the Office of the Assistant Secretary of Defense (OASD) for Command, Control, Communications, and Intelligence (C^3I), Information Systems, Room 3B252, Pentagon, Washington, DC 20301.

References are given in the order that the systems are listed in Tables 1 to 3.

System	Reference
ADEPT-50	C. Weissman, "Security Controls in the ADEPT-50 Time Sharing System," *Proceedings, 1969 AFIPS Fall Joint Computer Conference*, AFIPS Press, Arlington, Va., Vol. 35, pp. 119-133.
MULTICS	G. Organick, *The MULTICS System: An Examination of Its Structure*, MIT Press, Cambridge, Mass., 1972.
MULTICS Security Enhancements	J. Whitmore *et al.*, "Design for MULTICS Security Enhancements," ESD-TR-74-176, Air Force Electronic Systems Division, Dec. 1973. For information on security mechanisms available to users of the current commercially available MULTICS, see *MULTICS Programmers' Manual Reference Guide*, Honeywell #AG91, Revision 2, Honeywell Information Systems, Inc., Waltham, Mass., Mar. 1979.
MULTICS Security Kernel	W.L. Schiller, "Design and Abstract Specification of a MULTICS Security Kernel," MITRE ESD-TR-77-259, MITRE Corp., Bedford, Mass., Nov. 1977 (NTIS AD A048576).
MITRE Brassboard Kernel	W.L. Schiller, "Design of a Security Kernel for the PDP-11/45," MITRE MTP.-2709, MITRE Corp., Bedford, Mass., June 1973.
MITRE Secure UNIX	K. Biba, J. Woodward, and G. Nibaldi, "A Kernel Based Secure UNIX Design," MITRE ESD-TR-79-134, MITRE Corp., Bedford, Mass., May 1979.
UCLA Data Secure UNIX	G.J. Popek, M. Kampe, C.S. Kline, A. Stoughton, M. Urban, and E.J. Walton, "UCLA Secure UNIX," in *Proceedings, AFIPS National Computer Conference*, AFIPS Press, Arlington, Va., 1979, Vol. 48, pp. 355-364.

Military Message Experiment	S.H. Wilson, N.C. Goodwin, and E.H. Bersoff, "Military Message Experiment Final Report," NRL Memorandum Report 4456, Naval Research Laboratory, Washington, D.C., in press.
SHARE-7	"SHARE 7/Security Design, 1 February 1980," Fleet Combat Direction Systems Support Activity (FCDSSA), San Diego, CA 92147. (This is an informal document, not an officially published report.)
DAMOS	A. Hvidtfeldt and A. Smitt, "Manufacturers' Efforts in Computer Security: Christian Rövsing," in *Proceedings of the Fourth Seminar on the DOD Computer Security Initiative Program*, Aug. 1981, in press.
KVM/370	B.D. Gold, R.R. Linde, R.J. Peeler, M. Schaefer, J.F. Scheid, and P.D. Ward, "A Security Retrofit of VM/370," in *Proceedings, AFIPS National Computer Conference*, AFIPS Press, Arlington, Va., 1979, Vol. 48, pp. 335-342.
KSOS	E.J. McCauley and P.J. Drongowski, "KSOS: The Design of a Secure Operating System," in *Proceedings, AFIPS National Computer Conference*, AFIPS Press, Arlington, Va., 1979, Vol. 48, pp. 345-353.
SCOMP	C.H. Bonneau, "Secure Communications Processor Kernel Software, Detailed Specification, Part I, Rev. G," Honeywell Inc., Avionics Division, St. Petersburg, Fla., 1981.
GUARD (ACCAT, LSI)	D. Baldauf, "ACCAT GUARD Overview," MITRE MTR-3861, MITRE Corp., Bedford, Mass., Nov. 1979. S. Stahl, "LSI GUARD System Specification (Type A)," MITRE MTR-8452, MITRE Corp., Bedford, Mass., Oct. 1981.
GUARD (FORSCOM, RAP)	"FORSCOM Security Monitor Computer Program Development Specification Type B-5," Logicon, Inc., San Diego, Calif., Feb. 1981.
SDC Comm. Kernel	T. Golber, "The SDC Communication Kernel," in *Proceedings of the Fourth Seminar on the DOD Computer Security Initiative Program*, Aug. 1981, in press.
AUTODIN II	S. Bergman, "A System Description of AUTODIN II," MITRE MTR-5306, MITRE Corp., Bedford, Mass., May 1978. For a short summary of AUTODIN II, see I. Lieberman, "AUTODIN II: An Advanced Telecommunications System," Telecommunications 15 (5), 43-48 (May 1981).
SACDIN	"System Specification for SAC Digital Network (SACDIN)," ESD-MCV-1A, ITT Defense Communications Division, Nutley, New Jersey, 1978.
COS/NFE	S.R. Bunch, G. Grossman, and D.C. Healy, "Software Design Specification of the COS/NFE HUB Executive, Release 2," Digital Technology Incorporated, Champaign, Ill., Dec. 1980.
GNOSIS	"GNOSIS External Specification," 1st Ed., Tymshare, Inc., Cupertino, Calif., Mar. 1980.

PSOS	P.G. Neumann, R. Boyer, R.J. Feiertag, K. Levitt, and L. Robinson, "A Provably Secure Operating System: The System, its Applications, and Proofs," 2nd Ed., CSL-116, SRI International, Menlo Park, Calif., May 1980.
Message Flow Modulator	D.I. Good, "Message Flow Modulator Status Report, April, 1981," available from Naval Electronics Systems Command, Code 8144.
Secure Archival Storage System	R.R. Schell and L.A. Cox Jr., "A Secure Archival Storage System," in *Proceedings, FALL COMPCON*, Sept. 1980, pp. 679-682.
RAP GUARD	See GUARD, FORSCOM. MITRE is currently working on a specification for this system that will probably be similar to FORSCOM GUARD.
KAIS GUARD	M.H. Cheheyl, "KAIS Security Interface System Specification (Type A)," MITRE WP-23684, MITRE Corp., Bedford, Mass., July 8, 1981.
WWMCCS Local Network	L. Bernosky, "WWMCCS Information System (WIS) Computer Security," in *Proceedings of the Fourth Seminar on the DOD Computer Security Initiative Program*, Aug. 1981, in press.
NRL MMS/TDMS	C. Landwehr, "Security Model for a Military Message System," in *Proceedings of the Fourth Seminar on the DOD Computer Security Initiative Program*, Aug. 1981, in press.
MITRE DBMS Kernel	R.D. Graubart and J.P.L. Woodward, "A Naval Surveillance DBMS Security Model," MITRE MTR-8475, MITRE Corp., Bedford, Mass., Nov. 1981.
MITRE MLS Terminal	D. Solomon, "Processing Multilevel Secure Objects," in *Proceedings IEEE 1981 Symposium on Security and Privacy*, Oakland, Calif., Apr. 1981, pp. 56-61.
EIFEL 2	Lt. Col. Cerny, "ADP-Security Requirements for EIFEL 2," in *Proceedings of the Second Seminar on the DOD Computer Security Initiative Program*, Jan. 1980, pp. K-1 – K-32. (May be available from OASD (C^3I), Information Systems, Room 3E252, Pentagon.)
Canadian Maritime Command	No reference known to the author.